
MASSACHUSETTS
LIBRARIES
INSTITUTE of TECHNOLOGY

Inverse Spanning Tree Problems: Formulations and Algorithms

by
P.T. Sokkalingam
Ravindra K. Ahuja
James B. Orlin

WP #3890-96 April 1996

INVERSE SPANNING TREE PROBLEMS: FORMULATIONS AND ALGORITHMS

P. T. Sokkalingam
Department of Mathematics
Indian Institute of Technology,
Kanpur-208 016, INDIA

Ravindra K. Ahuja
Dept. of Industrial & Management Engg.
Indian Institute of Technology,
Kanpur-208 016, INDIA

James B. Orlin
Sloan School of Management,
Massachusetts Institute of Technology,
Cambridge, MA 02139, USA

(Revised March 26, 1996)

INVERSE SPANNING TREE PROBLEMS: FORMULATIONS AND ALGORITHMS

P. T. Sokkalingam*, Ravindra K. Ahuja**, and James B. Orlin***

ABSTRACT

Given a solution x^* and an a priori estimated cost vector c, the inverse optimization problem is to identify another cost vector d so that x^* is optimal with respect to the cost vector d and the deviation of d from c is minimum. In this paper, we consider the inverse spanning tree problem on an undirected graph G = (N, A) with n nodes and m arcs, and where the deviation between c and d is defined by the rectilinear distance between the two vectors (that is, L_1 norm). We show that the inverse spanning tree problem can be formulated as the dual of an assignment problem on a bipartite network $G^0 = (N^0, A^0)$ with $N^0 = N^1 \cup N^2$ and $A^0 \subseteq N^1 \times N^2$. The bipartite network satisfies the property that $|N^1| = (n - 1)$, $|N^2| = (m - n + 1)$, and $|A^0| = O(nm)$. In general, $|N^1| << |N^2|$. Using this special structure of the assignment problem, we develop a specific implementation of the successive shortest path algorithm that solves the inverse spanning tree problem in $O(n^3)$ time. We also consider the weighted version of the inverse spanning tree problem where we minimize the sum of the weighted deviations of arcs and show that it can be formulated as the dual of the transportation problem. Using a cost scaling algorithm, the transportation problem can be solved in $O(n^2 m \log(nC))$, where C denotes the largest arc cost in the data. Finally, we consider a minimax version of the inverse spanning tree problem and show that it can be solved in $O(n^2)$ time.

---

* Department of Mathematics, Indian Institute of Technology, Kanpur - 208 106, INDIA

** Dept. of Industrial & Management Engg., Indian Institute of Technology, Kanpur - 208 106, INDIA

*** Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

1. INTRODUCTION

Let $\mathcal{X}$ denote the set of feasible solutions of an optimization problem. Given a solution $x^* \in \mathcal{X}$ and an a priori estimated cost vector c, the inverse optimization problem is to identify another cost vector d so that $dx^* \leq dx$ for all $x \in \mathcal{X}$ and such that the deviation of d from c is minimum. Roughly speaking, the inverse optimization problem is to identify a cost vector d which is nearest to a specified cost vector c and with respect to which the given solution x^* is an optimal solution of the optimization problem. In this paper, we study inverse minimum spanning tree problems using three different ways to measure deviations between the cost vectors.

Bitran, Chandru, Sempolinski and Shapiro [1981] studied the inverse optimization for the capacitated plan location problem. To our knowledge, Bitran et al. introduced the concept of inverse optimization, although it was anticipated by Everett [1963] and by the literature in numerical analysis. Inverse network optimization problems were first studied by Burton and Toint [1992, 1994]. They studied the inverse multiple-source shortest path problem with deviation between two vectors c and d measured by the L_2 norm. They show applications of these problem in traffic modeling and seismic tomography, and suggest a nonlinear programming algorithm to solve the problem. Inverse minimum cost flow problems with L_1, L_2 and L_∞ norms have been studied by Sokkalingam [1995]. To the best of our understanding, prior to this research no one has studied the inverse spanning tree problems.

Inverse optimization is an alternative approach to measure deviation from optimality. Rather than measuring the distance of a solution x^* from optimality by comparing its objective value to the objective value to the optimum, one poses the following question: How much would one need to perturb the data so that x^* is optimum? This inverse perspective is of special interest when the cost data is subject to measurement error, which is typically the case. Also, the inverse optimization objective has the advantage that it is less sensitive to changes in the cost data. Changing a cost coefficient by one unit can have a major impact in deviation from optimality in the usual sense, but it can only increase the inverse objective function by 1 unit. We also note that the concept of *ε-optimality*, which is a critical aspect of the Goldberg-Tarjan [1987] algorithm for the minimum cost flow problem, is closely related to inverse optimization. One definition of ε-optimality is the following: a solution x^* is called *ε-optimal* for the minimum cost flow problem if it is possible to perturb each cost coefficient by at most ε so that x^* is optimal for the perturbed problem.

In this paper, we first study the inverse spanning tree problem in an undirected graph G = (N, A) with n nodes, m arcs, and in which the deviation between two arc cost vectors c and d is defined by the rectilinear distance between the two vectors. We show that the inverse spanning tree problem can be formulated as the dual of an assignment problem on a bipartite graph with $G^0 = (N^0, A^0)$ with $N^0 = N^1 \cup N^2$ and $A^0 \subseteq N^1 \times N^2$. The bipartite network G^0 satisfies the property that $|N^1| = (n - 1)$, $|N^2| = (m - n + 1)$,

and $|A^0| = O(nm)$. Often, $n << m$, hence $|N^1| << |N^2|$. Such an assignment problem is called *unbalanced*. We develop a specitic implementation of the well known successive shortest path algorithm for the assignment problem on G^0 which obtains a shortest path in an amortized (that is, average) time of $O(n^2)$. This running time is often much better than the running time of $O(|A^0|)$, which is the number of arcs in G^0. The resulting algorithm solves the assignment problem, and hence the inverse spanning tree problem in $O(n^3)$ time.

Next we consider the weighted version of the inverse spanning tree problem in which the deviation between two cost vectors c and d is defined by the weighted rectilinear distance between the two vectors. We show that the weighted spanning tree problem can be formulated as the dual of a transportation problem on G^0 and can be solved by a cost scaling algorithm in $O(n^2 m \log(nC))$ time, where C denotes the largest arc cost. Finally, we consider the minimax version of the inverse spanning tree problem and show that the minimax inverse spanning tree problem can be solved in $O(n^2)$ time.

This paper is organized as follows. Section 2 shows that the inverse spanning tree problem is the dual of an unbalanced assignment problem. Section 3 describes an algorithm for the assignment problem and shows how to obtain an optimal solution of the inverse spanning tree problem from the optimal solution of the assignment problem. Section 4 considers the weighted inverse spanning tree problem, and Section 5 studies the minimax version of the inverse spanning tree problem.

2. TRANSFORMATION TO THE ASSIGNMENT PROBLEM

In this section, we show that the inverse spanning tree problem can be transformed to an assignment problem. In this section, as well as elsewhere, we follow the network notation given in the book of Ahuja, Magnanti and Orlin [1993], and refer the reader to the same. We denote the complement of any set S by placing a bar on it, that is, by $\bar{S}$.

Let G = (N, A) be an undirected network consisting of the node set N and the arc set A. Let n = |N| and m = |A|. We assume that N = {1, 2, ... , n}, and A = $\{a_1, a_2, \ldots, a_m\}$. The data of the inverse spanning tree problem consists of a spanning tree T^* of G and an arc cost vector c with c_i denoting the cost of arc a_i. We assume without any loss of generality that $T^* = \{a_1, a_2, \ldots, a_{n-1}\}$. We refer to the arcs in T^* as *tree arcs* and the arcs in $\{a_n, a_{n+1}, \ldots, a_m\}$ as *nontree arcs*. The objective in the inverse spanning tree problem is to find an arc cost vector d such that T^* is optimal with respect to d and $\sum_{j=1}^{m} |c_j - d_j|$ is minimum.

In the given spanning tree T^*, there is a unique tree path between any two nodes; we denote by the set P_j the indices of the tree arcs on the path in T^* connecting the two endpoints of arc a_j. It is well known (see, for example, Ahuja, Magnanti and Orlin [1993]) that T^* is a minimum spanning tree with respect to the arc cost vector d if and only if

$d_i \le d_j$ for each $i \in P_j$ and for each j = n, ... , m. (1)

The inverse spanning tree problem can alternatively be conceived in the following manner. Consider the spanning tree T^*. The tree T^* may or may not be optimal with respect to the given cost vector c. If it is, then d = c is the desired cost vector with zero objective function value. If not, then we must perturb the given cost vector c by α so that T^* is optimal with respect to $(c + \alpha)$ and $\sum_{j=1}^{m} |\alpha_j|$ is minimum. This observation allows us to formulate the inverse spanning tree problem as the following mathematical program:

$$\text{Minimize } \sum_{j=1}^{m} |\alpha_j| \tag{2a}$$

subject to

$$c_i + \alpha_i \le c_j + \alpha_j \text{ for each } i \in P_j \text{ and for each } j = n, \dots, m, \tag{2b}$$

$$\alpha_j \text{ is unrestricted for each } j = 1, 2, \dots, m. \tag{2c}$$

Property 1. *There exists an optimal solution α of (2) in which $\alpha_i \le 0$ for each i = 1 to (n-1) and $\alpha_j \ge 0$ for each j = n to m.*

Proof. Observe that if $\alpha_i > 0$ for some i, $1 \le i \le (n-1)$, then we can set $\alpha_i = 0$ without violating any conditions in (2b) and without worsening the objective function value (2a). This establishes the first claim. A similar argument establishes the second claim. ♦

Using Property 1, we obtain the following equivalent formulation of the inverse spanning tree problem:

$$\text{Minimize } \sum_{j=n}^{m} \alpha_j - \sum_{i=1}^{(n-1)} \alpha_i \tag{3a}$$

subject to

$$c_i + \alpha_i \le c_j + \alpha_j \text{ for each } i \in P_j \text{ and for each } j = n, \dots, m, \tag{3b}$$

$$\alpha_i \le 0 \text{ for each } i = 1 \text{ to } (n-1), \text{ and } \alpha_j \ge 0 \text{ for each } j = n \text{ to } m. \tag{3c}$$

We now reformulate (3) using the concept of *Path Graph*, which allows us to express the constraints in (3b) in a manner more suitable for manipulation. The path graph, which we denote by $G^0 = (N^0, A^0)$ with $N^0 = N^1 \cup N^2$ satisfies $N^1 = \{1, 2, \dots, (n-1)\}$, $N^2 = \{n, n+1, \dots, m\}$, and $A^0 = \{(i, j) : i \in P_j, n \le j \le m\}$. Observe that A^0 contains an arc (i, j) for every inequality in (3b). (We may, however, exclude those arcs (i, j) for which $c_i \le c_j$ because any vector α satisfying (3c) will automatically satisfy $c_i + \alpha_i \le c_j + \alpha_j$.) We will

also like to restate (3) in the maximization form, we can do it by maximizing the negative of (3a). The modified formulation of the inverse spanning tree problem is as follows:

$$\text{Maximize} \quad \sum_{i \in N^1} \alpha_i - \sum_{j \in N^2} \alpha_j \tag{4a}$$

subject to

$$\alpha_i - \alpha_j \le f_{ij} \quad \text{for each arc } (i, j) \in A^0, \tag{4b}$$

$$\alpha_i \le 0 \text{ for each node } i \in N^1 \text{ and } \alpha_j \ge 0 \text{ for each node } j \in N^2, \tag{4c}$$

where $f_{ij} = c_j - c_i$ for each arc $(i, j) \in A^0$. The formulation (4) is a linear programming problem and has an associated dual. If we associate the dual variable x_{ij} with the arc (i, j) in (4b), then the dual of (4) can be stated as follows:

$$\text{Minimize} \sum_{(i,j) \in A^0} f_{ij} x_{ij} \tag{5a}$$

subject to

$$\sum_{\{j:(i,j) \in A^0\}} x_{ij} \le 1 \quad \text{for each node } i \in N^1, \tag{5b}$$

$$-\sum_{\{i:(i,j) \in A^0\}} x_{ij} \ge -1 \quad \text{for each node } j \in N^2, \tag{5c}$$

$$x_{ij} \ge 0 \quad \text{for each arc } (i, j) \in A^0. \tag{5d}$$

The formulation (5) is a variant of the well known assignment problem. In the standard assignment problem, the constraints (5b) and (5c) are in equality form and $|N^1| = |N^2|$. In our variant, it is possible that $|N^1| << |N^2|$. We refer to the formulation (5) as the *unbalanced assignment problem*.

3. SOLVING THE ASSIGNMENT PROBLEM

We have shown in Section 2 that the inverse minimum spanning tree problem can be transformed to an unbalanced assignment problem on a bipartite network $G^0 = (N^1 \cup N^2, A^0)$, where generally $|N^1| << |N^2|$. In this section, we develop a special purpose algorithm to solve the assignment problem, using the fact that $|N^1| << |N^2|$ to obtain a speedup. The algorithm runs in $O(n^3)$ time. We point out that for a dense network G (that is, $m = \Omega(n^2)$), the network G^0 may contain as many as $\Omega(n^3)$ arcs. Since any algorithm for the assignment problem must look at each arc at least one, our $O(n^3)$ algorithm for solving the assignment problem is an optimal algorithm for some classes of network G.

Consider any feasible assignment x of (5) which is a 0-1 vector. For this assignment, the nodes which satisfy the constraints (5b) or (5c) with equality are called *matched* nodes, and *unmatched* nodes

otherwise. Often $|N^1| < |N^2|$, and most nodes in N^2 are not be matched in an optimal assignment. As a matter of fact, in a minimum cost assignment, it is possible that some nodes in N^1 are not matched.

We can transform the unbalanced assignment problem (5) on $G^0 = (N^1 \cup N^2, A^0)$ into a minimum cost flow problem in the following manner. We introduce a source node s with supply b(s) = n, and add an arc (s, i) for each $i \in N^1$ with cost $f_{si} = 0$ and capacity $u_{si} = 1$. Similarly, we introduce a sink node t with b(t) = -n, and add an arc (j, t) for each $j \in N^2$ with cost $f_{jt} = 0$ and capacity $u_{jt} = 1$. We also add an arc (s, t) with $f_{st} = 0$ and $u_{ts} = (n+1)$. We set the supply/demand b(i) of each node $i \in N^0$ to zero. We set the capacity of each arc $(i, j) \in A^0$ to n; the cost of this arc is f_{ij}. Let $G' = (N', A')$ denote the resulting network. Let $n' = |N'|$ and $m' = |A'|$. Observe that $n' = (m+2)$ and $m' = O(nm)$. Also observe that similar to G^0, G' too is a bipartite network. We denote by $A'(i)$ the set of arc in A' emanating from node i. The following property establishes a connection between the minimum cost flow problem in G' and the assignment problem in G^0.

Property 2. *There is a one-to-one correspondence between feasible flows in G' and feasible assignments in G^0, and the cost of the flows and assignments are the same.*

Proof. Consider a feasible flow x in G'. Eliminating nodes s and t and the arcs incident on these nodes gives a solution of the assignment problem in G^0 having the same cost. Now consider a feasible assignment x in G^0. For each arc $(i, j) \in A^0$ with $x_{ij} = 1$, we send one unit of flow on arcs in (s, i) and (j, t). Then we send sufficient flow on arc (s, t) to satisfy the supply/demand constraints of nodes s and t. ♦

We can use the well known successive shortest path algorithm to solve the minimum cost flow problem in G' (see, for example, Ahuja, Magnanti and Orlin [1993] for a detailed description of this algorithm). The successive shortest path algorithm maintains a primal infeasible flow x and a dual feasible solution π satisfying the following optimality conditions:

(i) If $0 < x_{ij} < u_{ij}$, then $f_{ij}^{\pi} = 0$, (6a)

(ii) If $x_{ij} = 0$, then $f_{ij}^{\pi} \geq 0$, (6b)

(iii) If $x_{ij} = u_{ij}$, then $f_{ij}^{\pi} \leq 0$, (6c)

where $f_{ij}^{\pi} = f_{ij} - \pi(i) + \pi(j)$.

The successive shortest path algorithm starts with x = 0 and requires a dual solution π which together with x = 0 satisfies the optimality conditions given in (6). It is easy to verify that $\pi(j) = 0$ for all $j \in N^2 \cup \{t\}$, $\pi(i) = \min\{f_{ij} : (i, j) \in A^0\}$ for all $i \in N^1$, and $\pi(s) = \min\{\pi(i) : i \in N^1\}$ is one such solution. The successive shortest path algorithm proceeds by augmenting unit flow along shortest paths from node s to node t in the residual network $G'(x)$ defined with respect to the flow x. In general, the shortest paths are computed with respect to the reduced costs f^{π} which are always nonnegative. An iteration of the successive shortest path algorithm consists of determining shortest path distances d(i)'s from node s to all other nodes in the residual network $G'(x)$ with respect to the arc costs f_{ij}^{π}'s, updating node potentials as $\pi = \pi - d$, and

augmenting unit flow along the shortest path from node s to node t. After a number of shortest path augmentations, the arc (s, t) will become the shortest path from node s to node t, and the algorithm will terminate after augmenting flow on this arc. (Observe that arc (s,t) will become a shortest path because (i) b(s) = n, (ii) $|N^1| = (n-1)$, (iii) the capacity of each arc (s, i) is 1, which imply that any feasible flow will have positive flow on the arc (s, t).) Let x denote the optimal flow and π denote the optimal dual solution of the minimum cost flow problem.

Property 3. *The optimal primal solution x together with the optimal dual solution π of the minimum cost flow problem satisfies the following conditions:*

(a) $\pi(s) = 0$ *and* $\pi(t) = 0$.

(b) *For each arc (s, i), if* $x_{si} = 1$ *then* $\pi(i) \leq 0$; *if* $x_{si} = 0$ *then* $\pi(i) \geq 0$.

(c) *For some arc (j, t), if* $x_{jt} = 1$ *then* $\pi(j) \geq 0$; *if* $x_{jt} = 0$ *then* $\pi(j) \leq 0$.

Proof. We can assume without any loss of generality that $\pi(t) = 0$. Since $0 < x_{ts} < u_{ts}$, it follows from (6a) that $f^{\pi}_{st} = 0$ which implies $\pi(s) = 0$, establishing part (a) of the property. Now consider part (b). If $x_{si} = 1$ then it follows from (6c) that $f^{\pi}_{si} \leq 0$ which implies $\pi(i) \leq 0$. If $x_{si} = 0$, then it follows from (6b) that $f^{\pi}_{si} \geq 0$ which implies $\pi(i) \geq 0$. This establishes part (b). Next consider part (c). If $x_{jt} = 1$, then it follows from (6c) that $f^{\pi}_{jt} \leq 0$ which implies $\pi(j) \geq 0$. If $x_{jt} = 0$, then it follows from (6b) that $f^{\pi}_{jt} \geq 0$ which implies $\pi(j) \leq 0$, establishing part (c). ♦

We will now show that the solutions x and π can be used to obtain an optimal solution of the inverse spanning tree problem. It follows from Property 2 that the flow x has a corresponding assignment. For simplicity of notations, we view x as denoting the corresponding assignment as well.

Lemma 1. *The vector α defined by (7) is an optimal solution of the inverse spanning tree problem:*

$$\alpha_i = \begin{cases} \pi(i) & \text{for every matched node } i \text{ in } x \\ 0 & \text{for every unmatched node } i \text{ in } x. \end{cases} \qquad (7)$$

Proof. It follows from Property 3(b) and (c) that α is feasible to (4). Now consider a matched arc (i, j) . Since $0 < x_{ij} < u_{ij}$, it follows from (6a) that $f^{\pi}_{ij} = 0$, and thus $f_{ij} - \pi(i) + \pi(j) = 0$. Since both the nodes i and j are matched, using (7) we obtain that $\alpha_i - \alpha_j = f_{ij}$. Summing these equations for all matched arcs and using the facts that (i) $x_{ij} = 0$ for each unmatched arc (i, j), and (ii) $\alpha_i = 0$ for each unmatched node i yields

$$\Sigma_{i \in N1} \alpha_i - \Sigma_{j \in N2} \alpha_j = \Sigma_{(i,j) \in A0} f_{ij} x_{ij}.$$

Since α is a feasible solution of the primal problem (4), x is a feasible solution to the dual problem, and their objective function values are the same, it follows that α is an optimal solution of the primal problem (4). This completes the proof of the lemma. ♦

It follows from Lemma 1 that the optimal cost vector d of the inverse spanning tree problem is given by $d_j = c_j + \alpha_j$ for each j = 1, ... , m.

We now analyze the running time of the successive shortest path algorithm. The algorithm solves at most $|N^1|$ shortest path problems with arc lengths as f_{ij}^{π}'s which are nonnegative. Using Fibonacci heap implementation of Dijkstra's algorithm due to Fredman and Tarjan [1984], each shortest path problem can be solved in O(m' + n' log n') time. Consequently, this approach can solve the inverse spanning tree problem in $O(|N^1| (m' + n' \log n)) = O(n^2 m)$ time, because $|N^1| = (n-1)$, n' = (m+2), and m' = O(nm).

Improved Algorithm

The bottleneck step in our approach is the successive applications of Dijkstra's algorithm with each application requiring O(m' + n' log n) time. We will now describe an improved implementation of Dijkstra's algorithm for the case when $n^2 \leq (m' + n' \log n)$. In this case, we show that we can perform all the applications of Dijkstra's algorithm in $O(n^3)$ total time using data structures simpler than Fibonacci heap.

Dijkstra's algorithm, when applied on the residual network G'(x) with arc lengths f_{ij}^{π}'s, to find a shortest path from node s to node i, maintains a distance label d(i) with each node $i \in N'$. A distance label d(i) is either finite or infinite; if it is finite, then it denotes the length of some directed path from node s to node i; otherwise, it implies that a directed path to node i is yet to be discovered. A finite distance label d(i) is *permanent* if it is guaranteed to be the shortest path length to node i, and is temporary otherwise. In each iteration, Dijkstra's algorithm selects a minimum temporary distance label d(i), makes it permanent, and examines each arc $(i, j) \in A'(i)$ to update $d(j) = \min\{d(j), d(i) + f_{ij}^{\pi}\}$. The algorithm terminates when all distance labels become permanent.

We make two changes in Dijkstra's algorithm to improve its worst-case complexity when applied to G'.

Change 1. *Permanently label node t as soon as it has the minimum distance label among temporarily labeled nodes. Then terminate Dijkstra's algorithm.*

Recall that the successive shortest path algorithm requires a shortest path from node s to node t and such a path becomes available as soon as node t is permanently labeled. Hence there is no need to permanently label more nodes.

Recall from our previous discussion that in every iteration the successive shortest path algorithm updates the node potentials as $\pi(i) = \pi(i) - d(i)$ for each node i in N', which ensures that the reduced costs remain nonnegative. In case we terminate Dijkstra's algorithm prematurely, we need to update the node potentials as $\pi(i) = \pi(i) + \max\{d(t) - d(i)\}$ for each *permanently labeled* node i in N'. It follows from Lemma 2 proved below that updating the node potentials takes O(n) time and, as shown in Ahuja, Magnanti and Orlin [1993], it also ensures that the arc reduced costs remain nonnegative.

An immediate byproduct of Change 1 is the following lemma

Lemma 2. *Dijkstra's algorithm permanently labels at most 2n nodes*

Proof. In the residual network G'(x) nodes in N^1 , N^2 are either matched or unmatched. Since G'(x) permits at most $|N^1| = (n-1)$ units of flow to be sent from node s to node t, there will be at most (n - 1) matched nodes in N^1 and at most (n-1) matched nodes in N^2. Therefore, after at most 2(n - 1) nodes have been permanently labeled, an unmatched node in N^2, say node k, will be permanently labeled. Immediately thereafter, arc (k, t) is examined, which is the unique arc emanating from node k, and d(t) is updated to $d(t) = d(k) + f^{\pi}_{kt} = d(k)$ (because $f^{\pi}_{kt} = f_{kt} = 0$). Since in Dijkstra's algorithm, distance labels of permanently labeled nodes are non-decreasing, it follows that node t acquires the minimum distance label, and in the next iteration Dijkstra's algorithm permanently labels it. In total, the algorithm permanently labels at most 2n nodes. ♦

We now explain the second change. We partition the arc adjacency list A'(i) of each node $i \in N^1$ in G'(x) into two parts M(i) and U(i), where $M(i) = \{(i, j) \in A'(i) : \text{node j is matched}\}$ and $U(i) = \{(i, j) \in A'(i) : \text{node j is unmatched}\}$. Notice that $|M(i)| \le n$, but U(i) can be as big as $|N^2| - |N^1|$, which is O(m). In the second change, Dijkstra's algorithm refrains from examining all arcs in U(i).

Change 2. *In Dijkstra's algorithm, when node i is permanently labeled, then examine all arcs in M(i) but only the smallest cost arc in U(i).*

We now show that Change 2 does not affect the correctness of Dijkstra's algorithm. We first observe that $\pi(j) = 0$ for each unmatched node j in N^2. (This is true at initialization, and $\pi(j)$ is not updated until node j is matched.) So the arc in U(i) with the smallest cost is also the arc in U(i) with smallest reduced cost. Suppose that we apply Dijkstra's algorithm without Change 2; that is, we examine all arcs in U(i). Consider the first time an unmatched node $j \in N^2$ has the minimum temporary distance label. Clearly, $d(j) = d(i) + f^{\pi}_{ij}$ for some permanently labeled node $i \in N^1$ such that $(i, j) \in A^0$. We claim that arc (i, j) is the arc in U(i) with minimum reduced cost. For, if this is not true and (i, k) is the minimum reduced cost arc in U(i), then $d(k) = d(i) + f^{\pi}_{ik} < d(i) + f^{\pi}_{ij} = d(j)$, contradicting that node j is the minimum temporary distance label in N^2. This argument establishes that if instead of examining all arcs in U(i) we examine only the least reduced cost arc in U(i), Dijkstra's algorithm will run correctly.

A byproduct of Change 2 is the following lemma.

Lemma 3. *In any iteration, Dijkstra's algorithm will have at most 2n temporary distance labels.*

Proof. Each temporary distance label is *caused* by some permanently labeled node; we say that it was caused by the node which modified it last. Now notice that each node $i \in N^1$ is either temporarily labeled or permanently labeled; if it is permanently labeled then it can cause several temporary distance labels of matched nodes in N^2 but at most one unmatched node in N^2. Since there are at most (n - 1) matched nodes in N^2, there will be at most 2n temporary distance labels in any iteration. ♦

We can now analyze the worst-case complexity of the successive shortest path algorithm that uses Dijkstra's algorithm incorporating Change 1 and Change 2. It follows from Lemma 2 that Dijkstra's algorithm will perform O(n) iterations. It follows from Lemma 3 that in each iteration it can identify in O(n) time a node with the minimum temporary distance label, say node i. Examining all arcs in M(i) and the minimum cost arc in U(i) also takes O(n) time plus the time to identify the minimum cost arc in U(i). Thus each application of Dijkstra's algorithm takes $O(n^2)$ time plus the time needed to identify minimum cost arcs in U(i). Therefore, the successive shortest path algorithm, which applies Dijkstra's algorithm at most n times, runs in $O(n^3)$ time plus the time needed to identify minimum cost arcs in U(i).

We now explain how to identify minimum cost arcs in U(i) quickly. Recall that the arc in U(i) with the minimum reduced cost is also the arc in U(i) with minimum cost. So we maintain the arcs in U(i) for each node $i \in N^1$ using a binary heap with cost of the arc (i, j) as its key. These heaps are constructed once at the beginning, and are updated after each application of Dijkstra's algorithm, so that they can be reused in the next application. Initially, U(i) = A'(i) and these heaps for all the nodes can be constructed in O(m') time (see, for example, Cormen, Leiserson and Rivest [1990]). In a binary heap, each heap operation, like (i) identifying a minimum cost arc in U'(i); and (ii) deleting an arc in U'(i) can be performed in O(log n) time. Suppose during an application of Dijkstra's algorithm, an additional node in N^2, say node k, get matched. We then need to update some heaps. To do so, we consider each arc $(i, k) \in A^0$ with $i \in N^1$ and remove it from U(i); this takes a total of O(log n) time per arc and a total of O(n log n) time for node k. The total time taken by the heap operations in all the applications of Dijkstra's algorithm is $O(m' + n^2 \log n)$. Since m' = O(nm) and $m = O(n^2)$, the heap operations also take $O(n^3)$ time. We can summarize the discussion in this section by the following theorem.

Theorem 1. *The successive shortest path algorithm can solve the unbalanced assignment problem and, hence, the inverse spanning tree problem in $O(n^3)$ time.*

4. WEIGHTED INVERSE SPANNING TREE PROBLEM

In this section, we consider the weighted spanning tree problem. This problem is a generalization of the inverse spanning tree problem, where the objective is to minimize the weighted deviation from the given cost vector. Suppose we associate a nonnegative weight w_j with each arc $a_j \in A$. Then the weighted inverse spanning tree problem can be formulated as the following mathematical program:

Minimize $\sum_{j=n}^{m} w_j \alpha_j - \sum_{i=1}^{(n-1)} w_i \alpha_i$ subject to (3b) and (3c).

Using exactly the same method we used for the unit weight case, the weighted inverse spanning tree problem can be reformulated as

Maximize $\sum_{i \in N^1} w_i \alpha_i - \sum_{j \in N^2} w_j \alpha_j$ subject to (4b) and (4c).

The dual of this problem is as follows:

$$\text{Minimize } \sum_{(i,j)\in A^0} f_{ij} x_{ij} \tag{8a}$$

subject to

$$\sum_{\{j:\,(i,j)\in A^0\}} x_{ij} \le w_i \quad \text{for each node } i \in N^1, \tag{8b}$$

$$-\sum_{\{i:\,(i,j)\in A^0\}} x_{ij} \ge -w_j \quad \text{for each node } j \in N^2, \tag{8c}$$

$$x_{ij} \ge 0 \quad \text{for each arc } (i, j) \in A^0, \tag{8d}$$

where $f_{ij} = (c_j - c_i)$. The formulation (8) is a variant of the well known standard transportation problem. In the standard transportation problem, the constraints (8b) and (8c) are in the equality form and $\sum_{i\in N^1} w_i = \sum_{i\in N^2} w_j$, which are not satisfied in the variant. We refer to the formulation (8) as the *unbalanced transportation problem*. We will show that an adaptation for the cost scaling algorithm for bipartite networks can solve the unbalanced transportation problem in $O(n^2 m \log(nC))$ time, where $C = \max\{|c_j| : a_j \in A\}$.

The cost scaling algorithm requires that the mass balance constraints are in the equality form, which (8) does not satisfy. To satisfy this condition, we first construct the network G' = (N', A') as described in Section 3 with the modification that the capacity of the arc (s, i) with $i \in N^1$ is set to w_i and the capacity of the arc (j, t) with $j \in N^2$ is set to w_j. We set $b(s) = -b(t) = \min\{\sum_{i\in N^1} w_i, \sum_{j\in N^2} w_j\} + 1$. Also observe that the unbalanced transportation problem in (8) reduces to a minimum cost flow problem in G'.

Goldberg and Tarjan [1987] describe a cost scaling algorithm that can solve the minimum cost flow problem on the network G' = (N', A') in $O((n')^3 \log(n'C))$ time, where C is the largest magnitude of the arc costs. Ahuja, Orlin, Stein, and Tarjan [1994] describe an improved implementation of the cost scaling algorithm for those bipartite networks where one part is considerably smaller than the other part. This algorithm can solve the minimum cost flow problem in G' in $O((|N^1|)^3 + (|N^1| m') \log(nC))$. Since $|N^1| = (n - 1)$, $m' = O(nm)$, the running time of this algorithm becomes $O(n^2 m \log(nC))$.

Let x denote the optimal primal solution and π denote the optimal dual solution of the minimum cost flow problem. For these solutions, we obtain α in the following manner:

$$\alpha_i = \begin{cases} 0 & \text{if node } i \in N^1 \text{ and arc (s, i) is not saturated} \\ \pi(i) & \text{if node } i \in N^1 \text{ and arc (s, i) is saturated} \end{cases}$$

$$\alpha_j = \begin{cases} 0 & \text{if node } j \in N^2 \text{ and arc (j, t) is not saturated} \\ \pi(j) & \text{if node } j \in N^2 \text{ and arc (j, t) is saturated.} \end{cases}$$

In a manner analogous to the unweighted case, it can be shown that α defined in the above manner is optimal for the weighted inverse spanning tree problem. We have thus proved the following theorem.

Theorem 2. *The cost scaling algorithm for bipartite networks can solve the unbalanced transportation problem, and hence the weighted inverse spanning tree problem, in $O(n^2 m \log(nC))$ time.*

5. MINIMAX INVERSE SPANNING TREE PROBLEM

In this section, we consider the variation of the inverse spanning tree problem where the objective is to minimize the maximum arc deviation instead of minimizing the sum of the arc deviations. This problem can be formulated in the following manner:

$$\text{Minimize } [\max \{|\beta_j| : 1 \le j \le m\}] \tag{9a}$$

subject to

$$c_i + \alpha_i \le c_j + \alpha_j \text{ for each } i \in P_j \text{ and for each } j = n, \ldots, m,$$

or, alternatively,

$$\alpha_j - \alpha_i \ge c_i - c_j \text{ for each } i \in P_j \text{ and for each } j = n, \ldots, m, \tag{9b}$$

$$\alpha_i \le 0 \text{ for each } i = 1 \text{ to } (n-1), \text{ and } \alpha_j \ge 0 \text{ for each } j = n \text{ to } m. \tag{9c}$$

Let

$$\delta = \max [c_i - c_j : \text{for each } i \in P_j \text{ and for each } j = n, \ldots, m\}. \tag{10}$$

If $\delta < 0$, then c is an optimal cost vector for T^*. Thus, we consider the case when $\delta \ge 0$. It is easy to see that $\delta/2$ is a lower bound on the objective function value of the minimax inverse spanning tree problem, because for some arc pair $\alpha_j - \alpha_i \ge \delta$ and one of them will be at least $\delta/2$ in magnitude. It is also easy to see that $\alpha_i = -\delta/2$ for each i, $1 \le i \le (n-1)$, and $\alpha_j = \delta/2$ for each j, $n \le j \le m$, achieves this lower bound and satisfies every constraint in (9). We have established the following result.

Lemma 4. *$\alpha_i = \min\{0, -\delta/2\}$ for each i, $1 \le i \le (n-1)$, and $\alpha_j = \max\{0, \delta/2\}$ for each j, $n \le j \le m$, with δ defined by (10) is an optimal solution of the minimax inverse spanning tree problem.*

We now study the time complexity of the minimax inverse spanning tree problem. The computation of δ is the bottleneck operation in the algorithm. If done in an straightforward fashion, it takes O(mn) time. However, we can do it in $O(n^2)$ time as described next. Let $\mathcal{E} = \{e_{ij}\}$ denote an nxn matrix whose ij-th element gives the largest cost of a tree arc in the unique path in T^* connecting nodes i and j. We can determine the i-th row of $\mathcal{E}$ in O(n) time by performing a search of T^* starting at node i. Therefore, the $\mathcal{E}$ matrix can be computed in $O(n^2)$ time. We claim that

$$\delta = \max [e_{t[j],h[j]} - c_j : n \le j \le m], \tag{11}$$

where t[j] and h[j] respectively denote the tail and head nodes of the arc a_j. The claim follows from the fact that for any nontree arc a_j the tree arc a_i with the largest value of c_i will attain the largest value of $c_i - c_j$

and $c_{[j],h[j]}$ gives this value. Once the matrix $\mathcal{L}$ is available, we can compute δ in O(m) time using (11). We have thus established the following theorem.

Theorem 3. *The minimax inverse spanning tree problem can be solved in $O(n^2)$ time.*

ACKNOWLEDGMENTS.

We gratefully acknowledge suppport from the Office of Naval Research under contract ONR N00014-96-1-0051 as well as a grant from the United Parcel Service.

REFERENCES

Ahuja, R. K., T. L. Magnanti, and J. B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Inc., NJ.

Ahuja, R. K., J. B. Orlin, C. Stein, and R. E. Tarjan. 1994. Improved algorithms for bipartite network flow problems. *SIAM Journal on Computing* **23**, 903-933.

Bitran, G., V. Chandru, D. Sempolinski, and J. Shapiro. 1981. Inverse optimization: an application to the capacitated plant location problem. *Management Science* **27**, 1120-1141.

Burton, D., and Ph. L. Toint. 1992. On an instance of the inverse shortest paths problem. *Mathematical Programming* **53**, 45-61.

Burton, D., and Ph. L. Toint. 1994. On the use of an inverse shortest paths algorithm for recovering linearly correlated costs. *Mathematical Programming* **63**, 1-22.

Cormen, T. H., C. L. Leiserson, and R. L. Rivest. 1990. *Introduction to Algorithms.* MIT Press and McGraw-Hill, New York.

Dijkstra, E. 1959. A note on two problems in connexion with graphs. *Numeriche Mathematics* **1**, 269-271.

Everett, H. 1963. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research* **11**, 399-417.

Fredman, M. L., and R. E. Tarjan. 1984. Fibonacci heaps and their uses in improved network optimization algorithms. *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, pp. 338-346. Full paper in *Journal of ACM* **34** (1987), 596-615.

Goldberg, A. V., and R. E. Tarjan. 1987. Solving minimum cost flow problem by successive approximations. *Proceedings of the 19th ACM Symposium on the Theory of Computing*, pp. 7-18. Full paper in *Mathematics of Operations Research* **15** (1990), 430-466.

Murty, K. G. 1976. *Linear and Combinatorial Programming.* John Wiley & Sons, NY.

Sokkalingam, P.T., 1995. *The Minimum Cost Flow Problem : Primal Algorithms and Cost Perturbations.* Unpublished Dissertation, Department of Mathematics, Indian Institute of Technology, Kanpur, INDIA.